Fuzzy Rewriting
– Soft Program Semantics for Children –

Yasunori Harada
NTT Communication Science Laboratories,
NTT Corporation
hara@acm.org

Richard Potter
Japan Science and Technology
potter@is.s.u-tokyo.ac.jp

Abstract— **Rewriting systems are popular in end-user programming because complex behavior can be described with few or no abstractions or variables. However, rewriting systems have been limited to manipulating non-rotatable objects on a grid, such as in Agentsheets or Stagecast Creator. Systems that allow free-form movement of objects must use other techniques, such as the sequential programming by demonstration in Squeak. Viscuit is a new rewriting system that introduces fuzzy rewriting, which allows freely positioned and rotated objects to interact using only rewriting rules. The result is a system that allows users to specify animations in a highly interactive way, without textual language or menu selections.**

Keywords— **Visibility, Rewriting Systems, Rule-based Visual Language**

I. Introduction

Animations are a major part of the Internet and are being created by more and more people. It typically requires programming-like activity, which can be frustrating to non-programmers who simply want to make their artistic creations move as desired. Simple techniques, like keyframe animation, can be tedious and produce static results. Rewriting systems allow one to create dynamic open-ended animations without programming. However, current systems are limited to animating objects on a fixed grid. Rotation is sometimes possible, but requires the user (or a professional) write a program. This puts many simple animations out of the reach of many end-users.

The main problem is that removing grids and allowing rotation gives each object a large number of possible positions and orientations. It is not practical to have a different rule for each orientation, so we propose a new rewrting mechnism, fuzzy rewriting[1], that combines two techniques.

1. fuzzy matching, which handles a range of relative distinces and angles; and
2. fuzzy generating, which infers a new (possibly unique) state that stays within the bounds of user intentions.

A similarity function for object relationships is defined. It is used during both matching and generating. The function should be designed according to end-users' cognition. In this paper, however, we don't discuss the end-user aspect. Our function of similarity is defined artificially and has many parameters to widely change behavior. The fuzzy rewriting mechanism is implemented in a system calld Viscuit, which allows freely positioned and rotated objects to interact using only rewriting rules.

[1]We don't use the word 'fuzzy' as a technical term.

This paper is organized as follows. In the next section we compare our work with others. In section III we describe the behavior of fuzzy rewriting. Viscuit is introduced in section IV and examples of its use are shown in section V. Section VI shows precise computing for fuzzy rewriting.

II. Related Works

AgentSheet[3] is an if-then rule-based visual language. It is suitible for simulation. In the condition part, several primitives, visual coditions or nonvisual conditions, can be used. The user can express object arrangements to express conditions in a functional programming manner. An object is located on a grid, so visual expressions are restricted. Kidsim (Cocoa, Stagecast Creator)[2] is a rewriting visual langauge for objects on a grid. An object has several appearances, which can be used for expressing an object's direction, state, and so on. A rule rewrites arrangements of objects with its appearance. Flash and Director, by Macromedia, enable animation of objects that can be rotated, positioned, and scaled. Motion is directed by keyframes and is scripted exactly. An animation is tightly controled by keyframes or algorithmically by scripting, so it is too difficult for our target end-users. BITPICT[5] and Visulan[7] are rewriting languages for bitmaps. They find bitmap patterns that are matched by a before-pattern of a rule and replace them with the after-patterns of the rules. Visulan has built-in patterns that express the mouse-button status. When the system knows the mouse-button status has changed, it changes the pattern into the corresponding built-in pattern. To write a program that interacts with a mouse, the user creates a normal rule that simply looks for the built-in pattern. BITPICT and Visulan use only bitmaps for data and programs. There is no hiding of information. Scott Kim defined this property as "visibility". His demonstration system, *VIEWPOINT* [9], combines a font editor, a word processer, and a keyboard layout manager. When a user types a key, the system copies a font pattern from the corresponding key on the keyboard layout into the cursor. Using this technique plus a few special rules, VIEWPOINT can function as a word processor with word wrap. ChemTrains[4] is a graph-rewriting visual language. When the system finds a graph pattern matching the before-pattern of a rule, it replaces it with the after-pattern of the rule. It is a powerful language because of the high flexibility and expressiveness of the graph representation.

0-7803-8225-0/03/$17.00 © 2003 IEEE

All the above systems except VIEWPOINT have two system modes: editing and running. Typically, using these systems involves writing programs, setting the initial state, runing, and stopping. On the other hand, Vispatch[10] does not distinguish between these modes.

Vispatch is a graph rewriting visual language. Each rule has an event object in a before-pattern and zero or more event objects in an after-pattern. When a user clicks on or drags on an object, rewriting is started. If an event object exists in the after-pattern of a fired rule, the system generates a new event that starts the next rewriting. Vispatch successfully achieves interactive rewriting. A rule in Vispatch is constructed as an object that can be rewritten by another Vispatch rule. This enables interactive reflection and makes a self-extensible graphics editor possible.

There has been much work on a motion generation. In [8], for example, motion is generated from examples.

III. Fuzzy Rewriting

Fuzzy rewriting is a new rewriting mechanism. Let's look at some examples. Fig. 1 is a simple rewriting rule. The horizontal arrow is a rule object that separates an object group into a before-pattern and an after-pattern. The left side of a rule object is a before-pattern (called the rule head), and the right side is an after-pattern (called the rule body). A explosion mark in a rule head expresses a mouse click event. The rule head in Fig. 1 includes two objects, a sun and a star, and one event. The rule body has the same two objects, only slightly rotated. This rule means that, when the sun is clicked, the sun and star rotate.

Fig. 1. Two objects of fuzzy rewriting

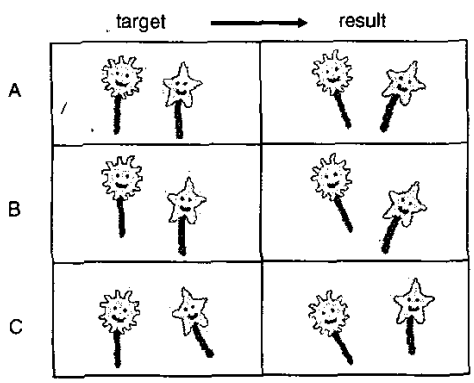

Fig. 2. Executions of Fig. 1

Fig. 2 shows three examples of rewriting with this rule. When the sun in the target-column is clicked, objects in the corresponding result-column replace objects in the corresponding target-column. In A, the arrangement of target-column objects is almost the same as rule-head objects, so the resulting arrangement is almost the same as the rule-body. In B, the star is lower than the sun, so the star in the result is also lower. In C, there is a deformation of arrangement, so the result is deformed.

Fig. 3 shows two objects whose positions are swapped so that they are opposite from those in the rule in Fig. 1. There are two possible results:[2] E preserves local constraints of the rule that keep rotation directions for each kind of object, so the star rotates clockwise and the sun counter-clockwise. F preserves the global constraints of the rule that decides rotation direction based on a relative position (not its kind), so a left-hand-side object rotates counter-clockwise and a right-hand-side object clockwise.

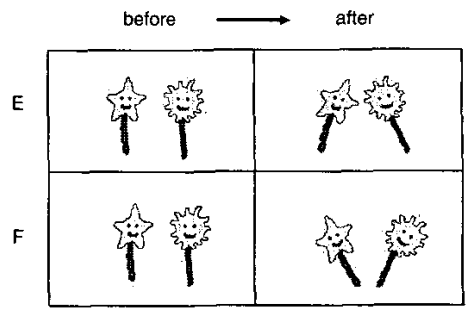

Fig. 3. Possible results of Fig. 1

Choosing the behaviour raises difficult problems. The preferences of end-users and the attractiveness of the result are important. The system needs consistency.

We give priority to local constraints, because our experience is that this produces more attractive results. They are also the easiest to implement. Our system therefore works like E (not F).

Let's compare a traditional rewriting system and a fuzzy rewriting one. Fig. 4 shows a data space to be rewritten and several rewritings on it, where a is a rewriting by a rule that has no variables, b is one by a rule that has variables, and c represents fuzzy rewriting. In a, a certain point (an input) is translated into another point (an output). In b, a certain area is translated into another certain area, and the correspondence between the input and the output area is defined by the rule. In c, like a, the rule has no variables. Points surrounding an input point are translated into other points surrounding an output point. Unlike b, input and output areas have no clear border.

[2]Of course, there is another possibility: the rules don't fire for different positions of objects like this. We can control this by changing the threshold for matching. This is discussed later.

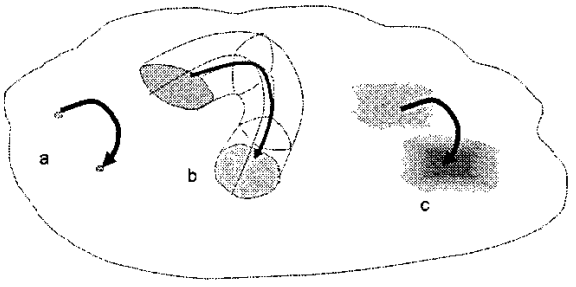

Fig. 4. Several Rewritings

IV. Viscuit

Viscuit is a new visual langauge that rewrites objects using the fuzzy techniques described in the previous section. In this section, we discuss how programs are written and run with Viscuit.

Viscuit is composed of paper and objects. A user can place several objects with free position and rotation (but no scaling) on a piece of paper. Three objects have special use: A *rule object* has two sets of objects, the head set and body set. A rule head includes one *event object* and a rule body includes zero or more event objects. An event object indicates where a click event is expected (in rule heads) or will be generated (in rule bodies). A *pointer object* refers to another piece of paper.

When a user clicks on a piece of paper, the system traces pointer objects on the paper recursively, and collects all rules from the traversed paper. Using the position of a click and the arrangement of objects on the clicked paper, the system selects the rule and the most similar arrangement of target objects. Afterthat the system rewrites objects according to the selected rule.

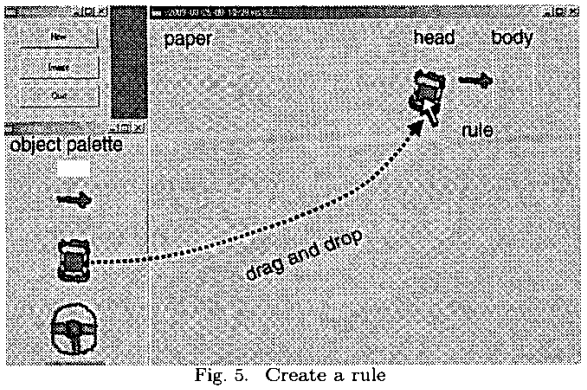

Fig. 5. Create a rule

Figs. 5-8 show user views. There is an object palette in the left of Fig. 5. To create a new object, a user drags the desired object from a palette and drops it into the target paper directly. A rule object captures its neighbor objects as rule head's or body's. After preparing a rule, and setting up an object to rewrite, the user clicks the object to see the rewriting result. In Fig. 6, the rule says when a car is clicked it moves forward, so this car (showing middle of the figure) moves forward by a user click.

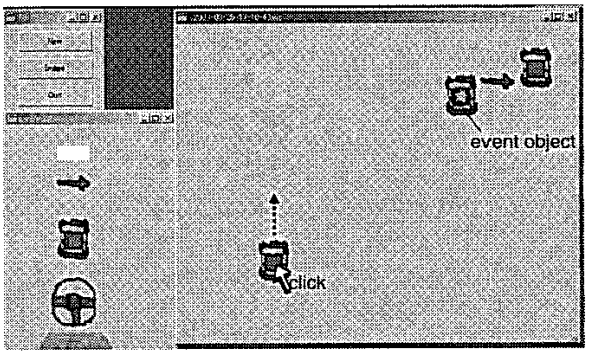

Fig. 6. Execute a program

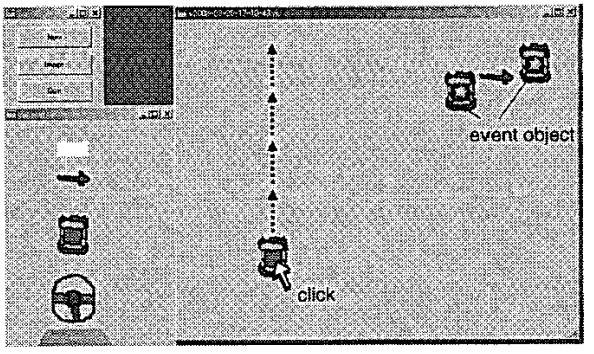

Fig. 7. Continuous rewriting

When a rule fires, an event object in a rule body genarates a click event and enqueues it into the event queue, where it behaves like an actual user click. When there is no user interaction, the system dequeues a posted event and tries to rewrite. In Fig. 7, the rule body has an event object. By clicking the target car, it moves upward continuously and disappears. A continuous rewriting

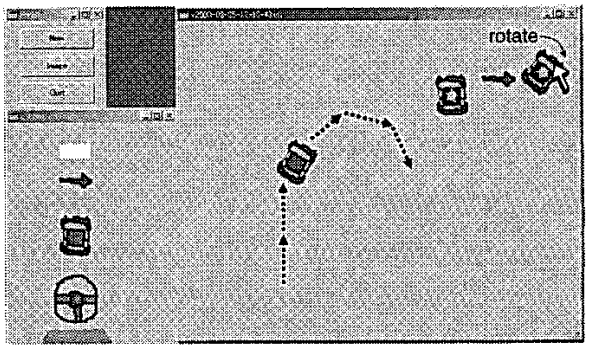

Fig. 8. Rotating a car

behaves a thread. Therefore, if there are several cars and they are each clicked by the user, then they move simultaneously.

Viscuit lets the user modify rules anytime. In Fig. 8, the user rotates the car in the rule body while the target car moves straight. After modifying the rule, the target car turns. The user can drive the car by modifying rules.

V. Execution Examples

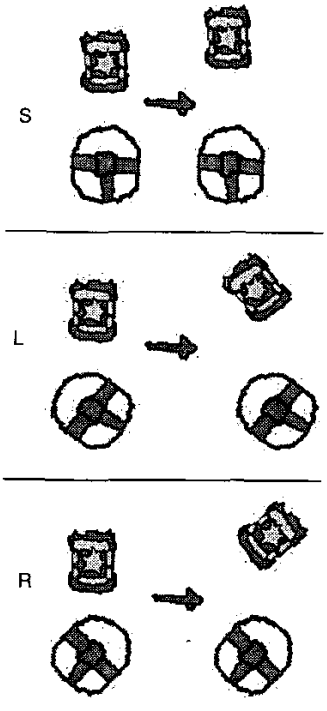

Fig. 9. Rules for driving (A)

Figs. 9 and 10 are two sets of rules that describe how a car should turn for a certain steering-wheel position. The difference between them is whether each rule includes a stand or not. In Fig. 9, the car wants to move to the same absolute direction as the steering wheel. When the car heads upward and the steering wheel is turned toward the right, the car turns right because rule R is fired. Now that both the car and the steering wheel are pointing in the same direction, rule S will fire and the car will go straight (Fig. 11, line A).

On the other hand, when the rules in Fig. 10 are applied to the target in Fig. 11, rule R in Fig. 10 is always fired. So the car turns always (Fig. 11, line B). If the steering wheel turns left, the car always turns left by rule L. If the steering wheel is straightened out, rule S would always fire and the car would always go straight.

The difference in these actions depends on the importance assigned to each relationship. In Fig. 9, there is

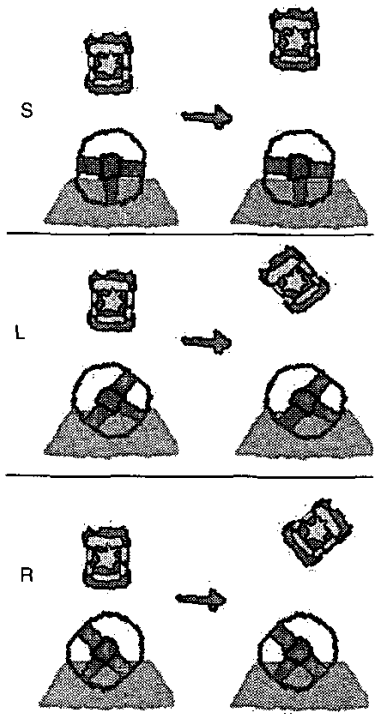

Fig. 10. Rules for driving with a stand (B)

only one relationship, which is the relative angle between the steering-wheel heading and the car heading. The car direction therefore affects the rule selection every time. On the other hand, in Fig. 10, the stand overlaps the steering wheel. Ovarlapped objects' relationships make higher similarity. Their relationships are assigned a higher importance than the car/steering-wheel relationship. Therefore, the rule is selected based on the relative angle between steering-wheel and stand.

Fig. 12 shows a single rule that animates soccer players kicking a ball. Its meaning is that when a soccer ball gets near the soccer player's foot, the ball should be moved out and in front of the player's head. In Fig. 13, for each click by a user or the system on the ball, the soccer player nearest the ball is selected, and the ball moves close to the foot of the next soccer player. In the resulting animation, the ball rotates clockwise like a soccer pass.

Fig. 14 only has one soccer player, but still produces a continuious animation because after the rule fires, the ball is still close enough to the soccer players foot to make the rule fire again. When a user clicks, the ball bounces around the player's head. This is good because the system never gets in a state totally unlike any of the body patterns. Therefore, while the system is unpredictable at the fine-grain level, its overall behaviour can be predicted from the rules.

Fig. 15 is a rule that shows a pass between two players.

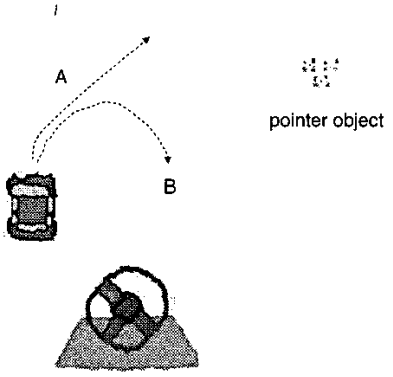

Fig. 11. Execution of driving

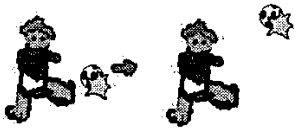

Fig. 12. Rule of soccer's pass

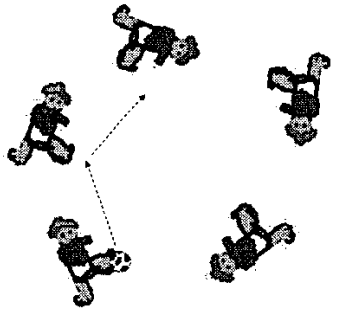

Fig. 13. Animation of soccer's pass

Fig. 14. Single play

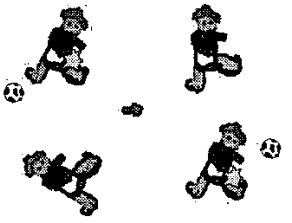

Fig. 15. Rule of soccer play

For each click on some player A, another player B is selected by object arrangement and the rule fires. This makes the ball move to B, and the system clicks B. Both A and B also move a small distance because they are shifted in the rule body pattern. The result is that players move about and seem to be passing the ball. Fig. 16 shows a snapsnot of the game in mid-play.

VI. Matching and Generating Objects

In implementing Viscuit, our strategy is as follows:

1. Define a function *rel2* that computes the arrangement similarity between a pair of objects and another pair of objects.
2. Use *rel2* to define a function *rel* that computes the arrangement similarity between one group of objects and another group of objects.
3. Select the rule and its mapping between head objects and target objects that maximize the value of *rel*.
4. Fire the rule if normarized *rel* is higher than the threshold.
5. Remove and generate objects whose arrangement maximizes the value of *rel*.

We define an object as having four attributes: *kind*, x, y, and *direction*, where *kind* is the kind of object, x and y are real numbers that express a center position of the object, and *direction* is a real number between -180 and 180 that expresses the screen direction of the object.

The distance between the center of an object P and the center of an object Q is $|PQ|$, the relative direction from P to Q is $rdir(P, Q)$, and the difference between the heading of P and the heading of Q is $angle(P, Q)$ (See Fig. 17).

The function $rel2(A, B, X, Y)$, which computes the similarity between relationship A and B and relationship X and Y is defined as

$$\begin{aligned} rel2(A,B,X,Y) \;=\; & C_0\delta(|AB|,|XY|,W_0) \\ & +\xi C_1\delta(rdir(A,B),rdir(X,Y),W_1) \\ & +\xi C_2\delta(rdir(B,A),rdir(Y,X),W_2) \\ & +C_3\delta(angle(A,B),angle(X,Y),W_3) \end{aligned}$$

where difference δ and weight ξ are

$$\delta(X,Y,W) \;=\; e^{-(X-Y)^2/W}$$

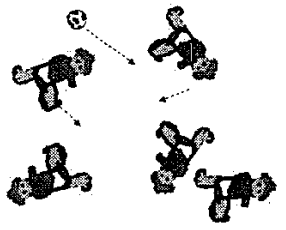

Fig. 16. Animation of soccer play

$$\xi = 1 - e^{-\frac{C_4}{(|AB|+|XY|+c)^2}} \quad (1)$$

The $\delta(X, Y, W)$ becomes 1 if X and Y are the same, otherwise it is close to 0. ξ becomes 0 if A and B have the same position and X and Y have the same position, otherwise it is close to 1. Parameters C_j and W_i are constants for tuning the system behavior.

The first term of *rel2* is a value showing how close distance $|AB|$ is to distance $|XY|$. The second term is one showing how close the relative direction $rdir(A, B)$ is to the relative direction $rdir(X, Y)$. The third term is one showing how close the relative direction $rdir(B, A)$ is to the relative direction $rdir(Y, X)$. $rdir(A, B)$ is unstable if A and B are very close. Weight ξ is therefore multiplied in the second and third terms to stabilize *rel2* behavior. The fourth term is a value showing how close $angle(A, B)$ is to $angle(X, Y)$.

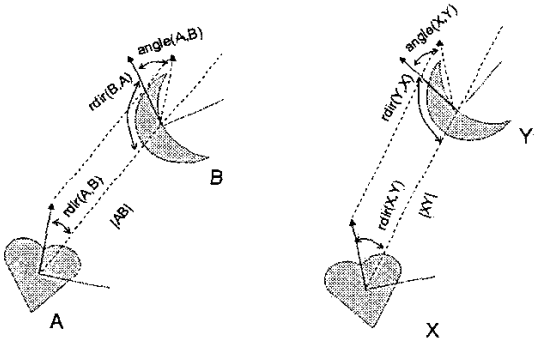

Fig. 17. Similarity between two pairs

Using $rel2(A, B, X, Y)$, we define function *rel*, which computes the similarity between an object group A and another object group that is defined by mapping function *map* as

$$rel(A, map) = \sum_{\substack{i \in A, j \in A \\ i \neq j}} w(i, j) rel2(i, j, map(i), map(j))$$

where $w(i, j)$ is a weight whose value changes according to whether object i and object j overlap or not. If they do, w has a higher value. This means ovarlapped objects get priority over other relationships.

Fig. 18 shows an example of a fuzzy rewriting. There is a rule that includes head objects (a, b, c), body objects (A, B, C), and an event object on object a of the head. When a user clicks on object 2, the system is activated. The system tries to match head objects and several objects on the target. Here, let some mapping *mat* be $2 = mat(a)$, $3 = mat(b)$ and $4 = mat(c)$. The value of $rel(\{a, b, c\}, mat)$, called the matching value of the rule, is computed by

$$\begin{aligned} rel(\{a, b, c\}, mat) &= w(a, b) rel2(a, b, 2, 3) \\ &\quad + w(b, c) rel2(b, c, 3, 4) \\ &\quad + w(c, a) rel2(c, a, 4, 2). \end{aligned}$$

The system looks for a mapping that maxmizes the matching value of this rule. Let this matching value be the maxmun matching value (MMV) of the rule and this mapping be the maxmun mapping of the rule. For each available rule, the system selects one rule that has the maxmun MMV. Whether the selected rule is fired or not depends on how similar the relationships are. MMV is normalized by percentage. A normalized MMV of 100% means the rule and the target have the same relationship exactly. If the normalized MMV of the selected rule is higher than the pre-defined threshold, the rule is fired.

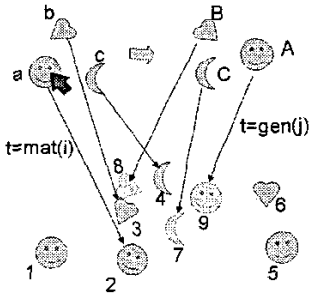

Fig. 18. Matching and generating

After the rule is fired, objects corresponding to rule head objects $\{2, 3, 4\}$ are deleted and other objects corresponding to rule body objects $\{7, 8, 9\}$ are generated. Let's denote the mapping *gen* corresponding to rule body objects and generated objects as $8 = gen(B)$, $9 = gen(A)$ and $7 = gen(C)$.

Arrangements of generated objects are computed by maximizing the following expression:

$$G(H,B,mat,gen) = \sum_{\substack{i\in H\\ j\in B\\ i\neq j}} w(i,j)rel2(i,j,mat(i),gen(j)) + rel(B,gen)$$

This means the first term computes a value showing how similar the relationships of head-body objects are to those of deleted/generated objects. In this example, the first term is

$$w(a,A)rel2(a,A,2,9) + w(b,B)rel2(b,B,\acute{3},8) + w(c,C)rel2(c,C,4,7).$$

The second term computes a value showing how similar the relationships of body objects are to those of generated objects. This is the reason for the swinging ball animation in Fig. 14, there are opposite effects (a ball go upward or downward) from the first and second terms.

To simplyfy computing, if all attributes of a head object and a body are the same. The system doesn't touch it (i.e., dowsn't delete and generate). A user interface support exists for this. When the user modifies a rule, a body object motion is snapped according to head objects location and angle.

VII. Consideration

Viscuit inherits features of rewriting language, so it has the basic mechanisms of computing. A sequence of click events has thread behaviour, as already mentioned. Viscuit also has rule inheritance because object patterns can express inclusion relationships. For example, each rule in Fig. 9 includes a rule in Fig. 10. If we use these rule sets simultaneously, the rules of Fig. 10 are used when the steering-wheel and stand overlap, and those of Fig. 9 are used otherwise.

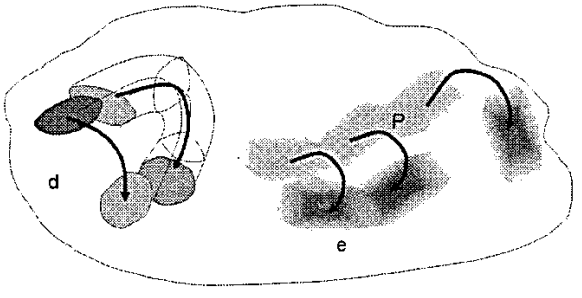

Fig. 19. Rewriting with several rules

Fig. 19 shows rewritings with several rules. In traditional rewriting, sometimes input areas of several rules with variables overlap like in d. In such a case, prioritizing the rules allows the system to control which rules should be fired. In this way, a complement of input areas can be expressed. On the other hand, in a fuzzy rewriting system, only fuzzy areas are expressed for each rule, so such a complement is difficult to express.

To express a complicated relationship by fuzzy rewriting, several rules (pairs of point) are given, like in e. These behave like a rule with variables (in Fig. 4b). This process is called Programming by Example (PBE). In a PBE system, if there are insufficient examples to generate rules (program), the system cannot proceed. However, in a fuzzy rewriting system, it is okay to do something, but the result becomes vague.

In Fig. 10, although the steering-wheel direction is matched fuzzily, the generated car animation moves in only one of three discrete ways: straight, curving at a set radius to the right, or curving at a set radius to the left. On might ask how we could have the steering-wheel control the car's turn in fine increments. One area for future research is to consider how to merge rules when more than one has high similarity. This would allow linear approximations to be expressed, which would create smooth intermediate behaviours. This is an example of Fig. 19e.

The threshold of the fire ratio can be adjusted by the user. The user repeatedly issues an event but no rule fires, and the system automatically lowers the threshold until some rule will fire. On the other hand, if many rewritings occur whose ratio is much higher than the current threshold, then the threshold is automatically raised to avoid unwanted rewriting. An event generated by the system doesn't affect the threshold adjustment.

An informal study of 40 children between age 6 and 10 was performed. The children were happy playing with the system for 30 minutes. In a post study interview, over 90% of the children said the system was interesting. They had no trouble creating rules, and experimented with them until they got the desired effect. However 25% of the children felt that Viscuit has user interface problems. We are improving the usability of the system based on this feedback. For example, Viscuit's technique for dragging objects allows both position and rotation to be changed with one mouse drag. Some children had trouble moving the objects without causing unintended rotation. This should be easy to fix by fine-tuning parameters that controlling such user interface behaviours.

VIII. Conclusion

We develop a new visual language, Viscuit, and its execution mechanism, fuzzy rewriting. Viscuit can treat an object as free-positioning and free-rotating. A rewriting rule is interpreted fuzzily, so a similar arrangement of objects can be rewritten as an appropriate arrangement. By continuous rewriting, Viscuit can express an animation whose local behavior is controlled by rules.

Demonstrations and the beta release of Viscuit can be

found in `http://www.viscuit.com` .

ACKNOWLEDGEMENT

We would like to thank Dr. Fusako Kusunoki and Miss Miyuki Kato of Tama Art University for their contribution to the visual design of Viscuit.

REFERENCES

[1] A. Baba and J. Tanaka : Eviss : a Visual System Having a Spatial Parser Generator, IPSJ Journal Vol.39 No.05. 023.

[2] A. Cypher, and D.C. Smith : KidSim: End User Programming of Simulations, CHI' 95.

[3] A. Repenning, and J.Ambach, Tactile Programming : A Unified Manipulation Paradigm Supporting Comprehension, Composition and Sharing, VL'96.

[4] B.Bell, and C.Lewis : ChemTrains: A Languages for Creating Behaving Pictures, VL' 93.

[5] G.W. Furnas : New Graphical Reasoning Models for Understanding Graphics Interfaces, CHI'95.

[6] K. Kahn: ToonTalk - An Animated Programming Environment for Children, Journal of Visual Languages and Computing, pp.197-217, June, 1996.

[7] K. Yamamoto: 3D-Visulan: A 3D Programming Language for 3D Applications. Pacific Workshop on Distributed Multimedia Systems (DMS96), pp.199-206, 1996.

[8] O. Arikan and D. A. Forsyth : *Interactive Motion Generation from Examples*, Proceedings of the 29th annual conference on Computer graphics and interactive techniques, 2002.

[9] S. Kim : Viewpoint : Toward a Computer for Visual Thinkers, Stanford University, 1988.

[10] Y. Harada, K. Miyamoto, R.Onai: VISPATCH: Graphical rule-based language controlled by user event, VL'97.

[11] Stagecast Creator : `http://www.stagecast.com/` .